Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Специальное машиностроение
КАФЕДРА Подводные роботы и аппараты

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент _____Андреев Евгений Викторович
фамилия, имя, отчество

Группа СМ11-31М

Тип практики Эксплуатационная_____

Название предприятия НУК СМ МГТУ им. Н.Э. Баумана

Студент _____ Андреев Е.В.
подпись, дата *фамилия, и.о.*

Руководитель практики _____ Макашов А. А.
подпись, дата *фамилия, и.о.*

Оценка _____

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Задание на прохождение практики:

1. Прохождение вводного инструктажа и инструктажа по технике безопасности.
2. Изучение алгоритмов компенсации неравномерности освещённости Single Scale Retinex и Multi Scale Retinex и их улучшенных версий.
3. Реализация алгоритмов компенсации неравномерности освещения на языке Python.
4. Сравнение алгоритмов компенсации на тестовом видео.
5. Измерение потоков и задержек IP-камер ACE-AP40 и Beward BD3595Z33.
6. Составление отчёта о прохождение практики.

Студент _____ Андреев Е. В.
(Подпись, дата) (И.О.Фамилия)

Руководитель практики _____ Макашов А. А,
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4

1. Общая информация о предприятии 5

2. Исследование алгоритмов коррекции изображений с неравномерной освещённостью 7

3. Измерение потоков и задержек IP-камеры ACE-AP40 11

4. Измерение потоков и задержек IP-камеры Beward BD3595Z33 15

ЗАКЛЮЧЕНИЕ 20

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21

Приложение А. Листинг программы, использующей алгоритм SSR 22

Приложение Б. Листинг 2. Алгоритм MSR и его вариации 23

Приложение В. Скрипт, реализующий многопроцессорность 27

ВВЕДЕНИЕ

Цель производственной практики – ознакомиться с историей НИИСМ НУК СМ, с основами работы инженерно-конструкторского отдела, получить практический опыт.

Задачей практики было:

1. Изучение алгоритмов компенсации неравномерности освещённости Single Scale Retinex, Multi Scale Retinex и его улучшенных версий.
2. Реализация алгоритмов компенсации неравномерности освещения на языке Python.
3. Сравнение алгоритмов компенсации на тестовом видео.

1. Общая информация о предприятии

Научно-исследовательская практика проходила на базе отдела СМ4-2 «Подводные системы» Научно-исследовательского института специального машиностроения МГТУ им. Н. Э. Баумана. НИИСМ НУК СМ МГТУ им. Н.Э.Баумана решает следующие основные задачи:

- проведение фундаментальных, поисковых и прикладных научных исследований в специальных областях машиностроения, создание новых высокоэффективных технологических процессов и машин;
- выполнение полных циклов опытно-конструкторских и научно-исследовательских работ с созданием головных, экспериментальных и опытных образцов техники;
- участие в промышленном освоении и выпуске продукции машиностроения, приборостроения и товаров народного потребления;
- участие в подготовке инженеров и специалистов высшей квалификации путем привлечения студентов и аспирантов к договорным научно-исследовательским работам;
- участие профессорско-преподавательского состава НУК СМ на условиях совместительства и контракта в выполнении научно-исследовательских, опытно-конструкторских и внедренческих работ НИИ СМ.

Отдел «Подводные системы» (СМ4-2) НУК СМ МГТУ им. Н.Э. Баумана специализируется уже более 20 лет на разработке и исследовании подводных телеуправляемых аппаратов любого класса, включая все виды силовых систем, систем управления и обеспечивающих комплексов.

Преимущественная форма работ отдела – заказная разработка на ОКР с испытанием и сдачей заказчику действующего изделия. В ряде случаев предусматривается авторское сопровождение изделия в процессе эксплуатации.

К настоящему времени отдел СМ4-2 выполняет на современном уровне всех требований ОКР следующие разработки:

- рабочие или осмотровые телеуправляемые аппараты со всеми обеспечивающими и рабочими системами;
- электрические и гидравлические движительно-маневровые комплексы подводных аппаратов;
- многофункциональные электрогидравлические следящие манипуляторы;
- подводные силовые гидравлические системы;
- видеосистемы и светотехнические комплексы;
- системы локальной видеонавигации;
- системы управления пространственным движением и устройствами подводного аппарата;

стенды для наземной отработки информационно-измерительных комплексов подводных аппаратов.

2. Исследование алгоритмов коррекции изображений с неравномерной освещённостью

Алгоритмы Multi-Scale Retinex (MSR) и Single-Scale Retinex (SSR) основаны на теории ретинекса [1], направленной на объяснение восприятия цветов человеческим зрением. Благодаря данной теории появились алгоритмы, предназначенные для усиления локального контраста на изображении.

SSR [2] производит выравнивание освещённости, при этом сохраняя контраст в ярко и плохо освещённых областях. Формула данного алгоритма приведена ниже:

$$\mathrm{R}(x, y, sigma) = \log[I(x, y)] - \log[I(x, y)] \otimes G(x, y, sigma),$$

где

R – преобразованное изображение,

I – исходное изображение,

G – Гауссиан,

x, y – координаты конкретного пикселя,

sigma – коэффициент размытия,

$\otimes$ – оператор свёртки.

Однако напрямую применить формулу выше можно только к одноканальному (в оттенках серого) изображению. Для трёхканального алгоритм логичнее всего применять к каналу яркости после преобразования исходного изображения в цветовое пространство HSV [3], где

H – hue, цветовой тон;

S – saturation, насыщенность;

V – value, значение цвета или же яркость.

В качестве тестового изображения возьмём рисунок 1.

Рисунок 1 – Тестовое изображение

Результат преобразования показан на рисунке 2.

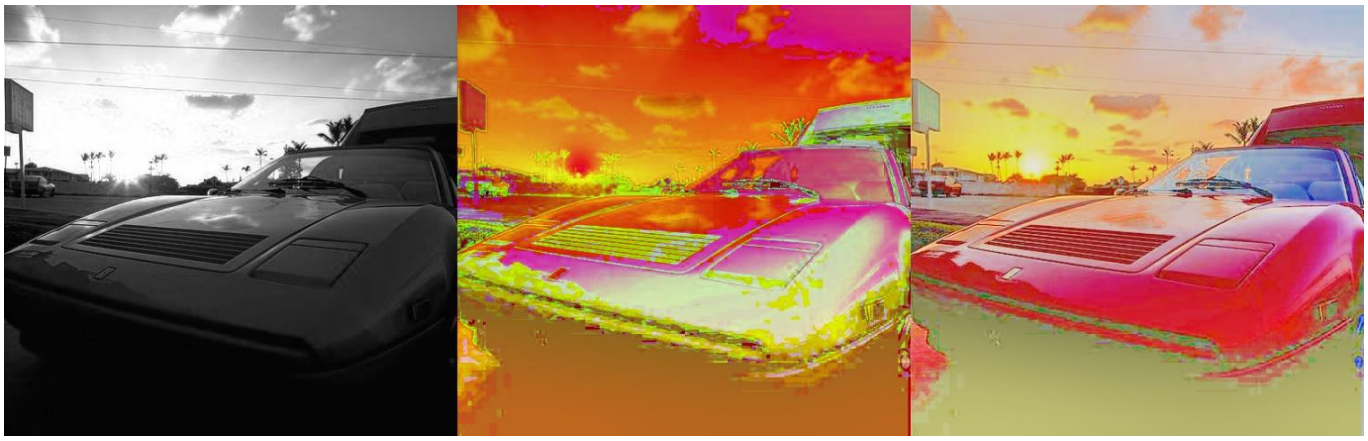


Рисунок 2 – Демонстрация SSR для трёхканального изображения. Слева направо: исходное изображение в градациях серого, преобразование в пространство HSV, скорректированное изображение

Листинг программы приведён в приложении А.

Алгоритм SSR не всегда даёт качественный результат. В свою очередь MSR [2] способен обеспечить приемлемый баланс между полученной цветопередачей и корректировкой освещённости. Результат работы алгоритма определяется как взвешенная сумма SSR с различными коэффициентами, которые в сумме должны давать единицу:

$$R_{MSR} = w_1 \cdot SSR_1 + w_2 \cdot SSR_2 + ... + w_N \cdot SSR_N,$$

где

R_{MSR} – выходное изображение,

w_i – весовые коэффициенты,

N – количество каналов. Чаще всего равно 3.

Рисунок 3 – Демонстрация различных вариантов алгоритма MSR. Слева направо: MSR с цветокоррекцией по пороговым значениям, MSRCR (Color Restoration) с восстановлением цвета, MSRCP (Chromatic Preservation) с предохранением цвета

Для полноты исследования сравнение также было проведено на кадрах подводной съёмки, полученных во время извлечения квадрокоптера со дна озера. Видеозапись взята из открытого источника. Результат показан на рисунке 4.

Рисунок 4 – Сравнение на кадре из подводного видео. Слева направо: ориганальное изображение, MSRCP, SSR

Многомерный Retinex позволяет получить большую чёткость и контраст деталей на изображении с неравномерной освещённостью, однако он требует существенно большей вычислительной мощности. В таблице 1 приведено сравнение затраченного на преобразование времени двух исследуемых алгоритмов, реализованных согласно листингам в приложениях А и Б.

Таблица 1. Сравнение времени преобразования

MSRCP	SSR
581 мин = 9,7 ч	25 мин

Для ускорения расчётов был написан сценарий, использующий преимущества наличия нескольких ядер в современных компьютерах. Лигтинг приведён в приложении В.

3. Измерение потоков и задержек IP-камеры ACE-AP40

Миниатюрная IP-камера ACE-AP40 [4] предназначена для видеонаблюдения в режиме реального времени. Подключение производится по протоколу ONVIF с помощью утилиты ONVIF Device Manager. Параметры по умочанию показаны на рисунке 5.

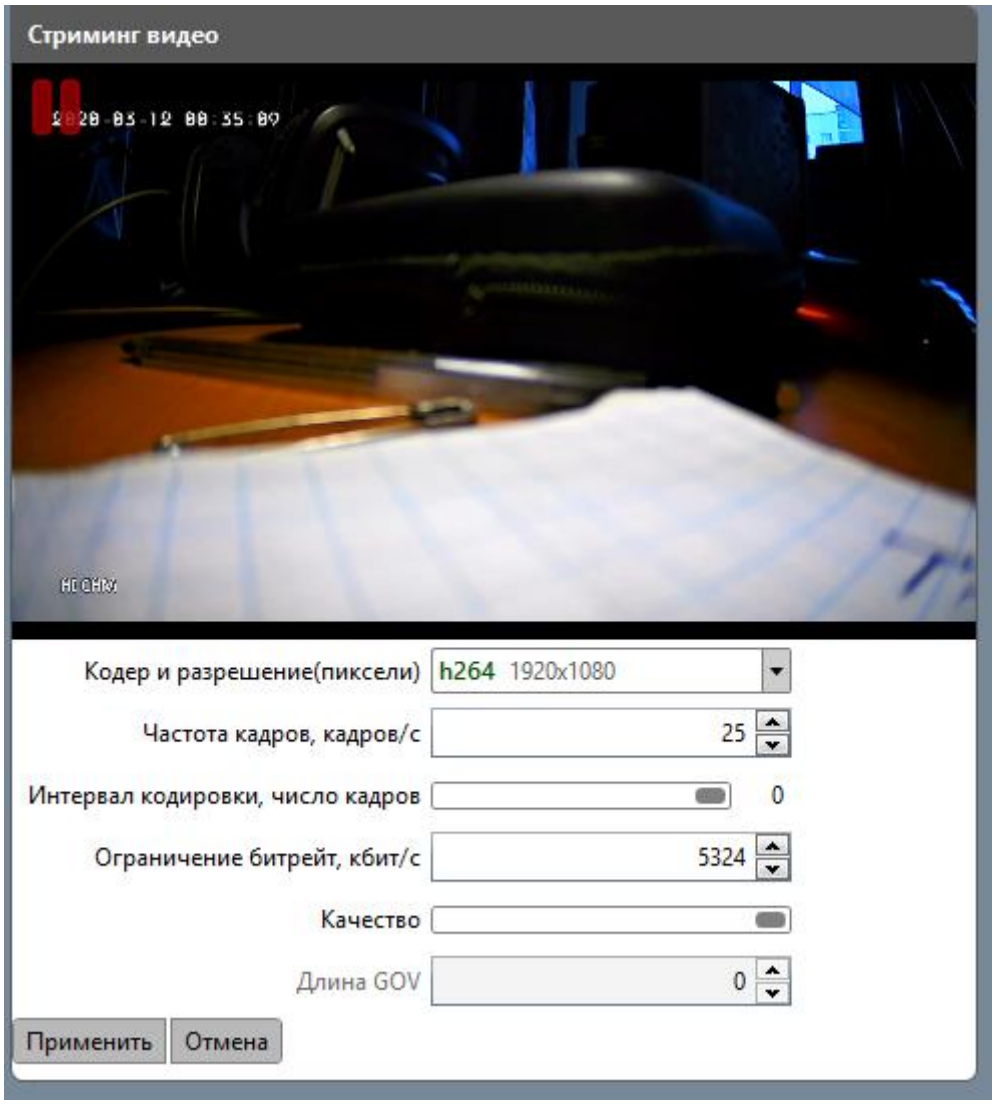


Рисунок 5 – Параметры камеры по умолчанию

Дальнейшее измерение потоков и задержек производилось при разрешении 1920 на 1080 пикселей.

Измерение потоков:

- минимум 1,9 Мбит/с

- максимум: в пике до 7,7 Мбит/с

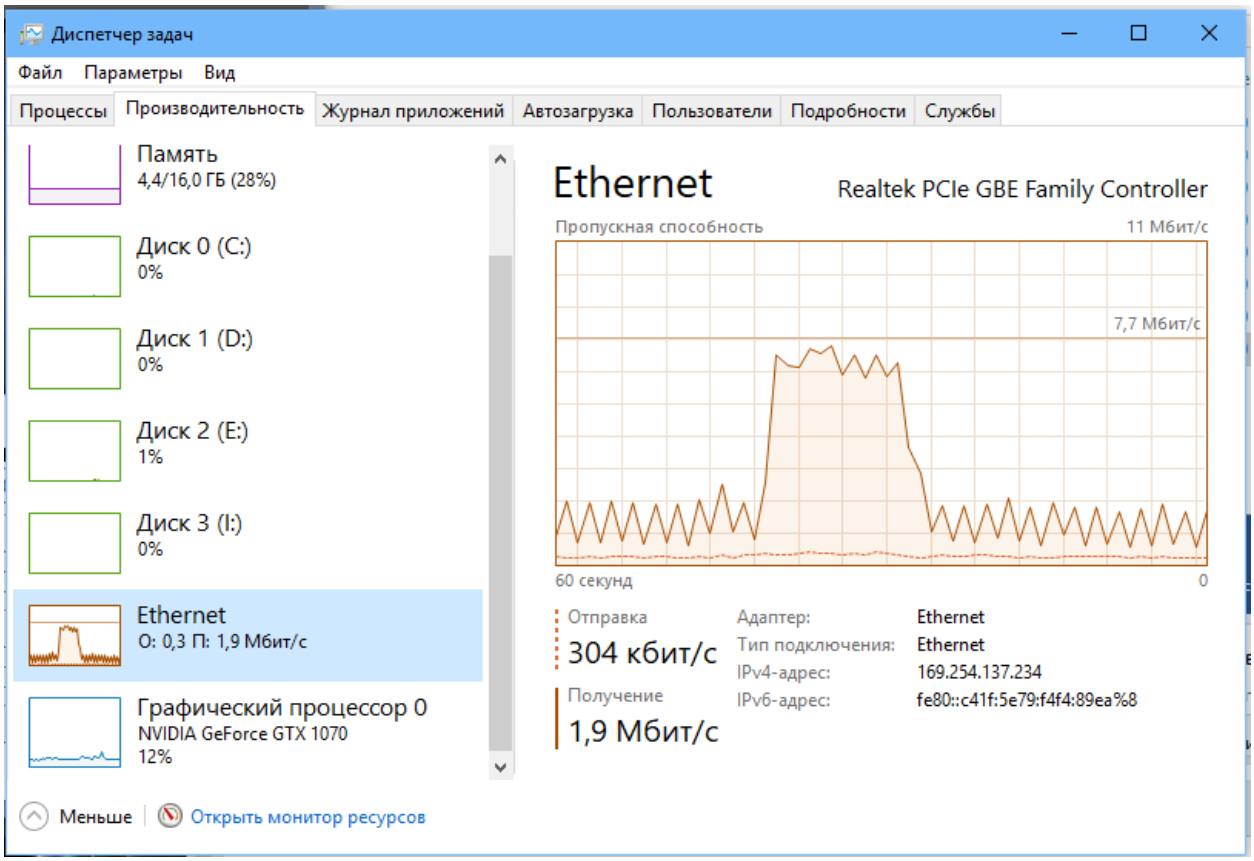
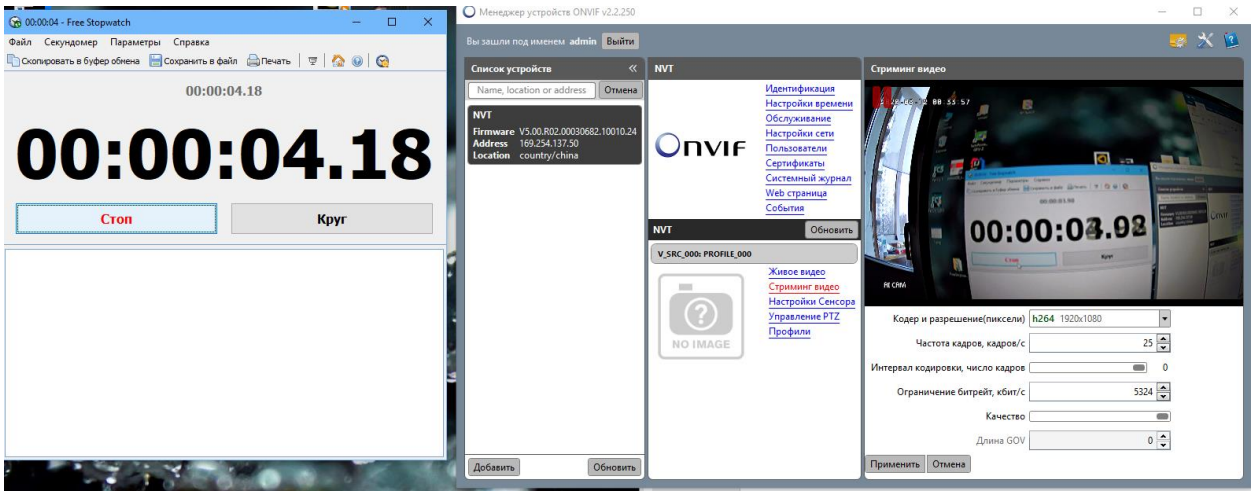


Рисунок 6 – Снимок экрана диспетчера задач при подключённой камере

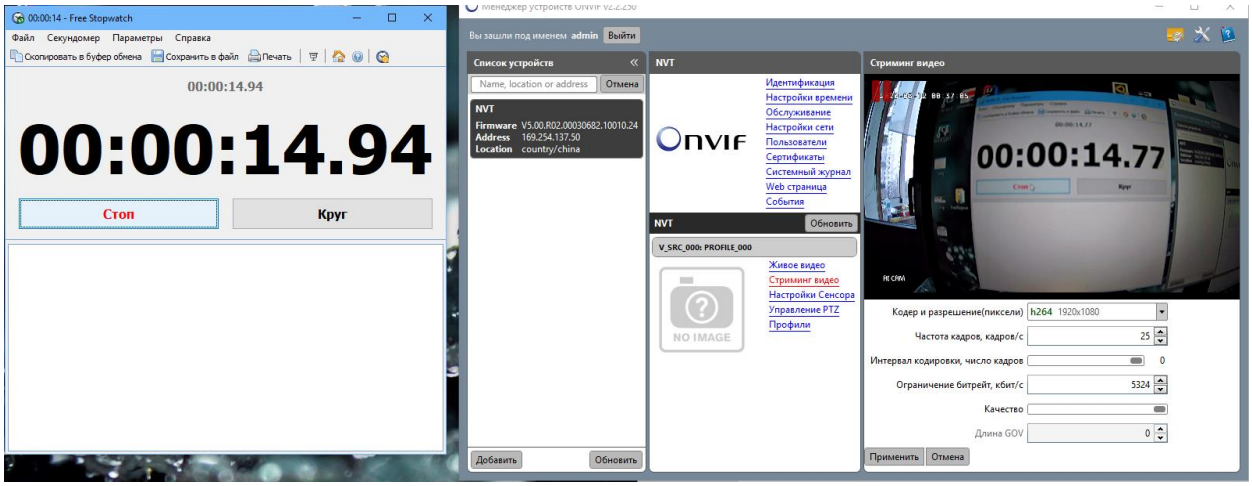
Измерение задержек

Снимок экрана с параметрами по умолчанию показан на рисунке 5.

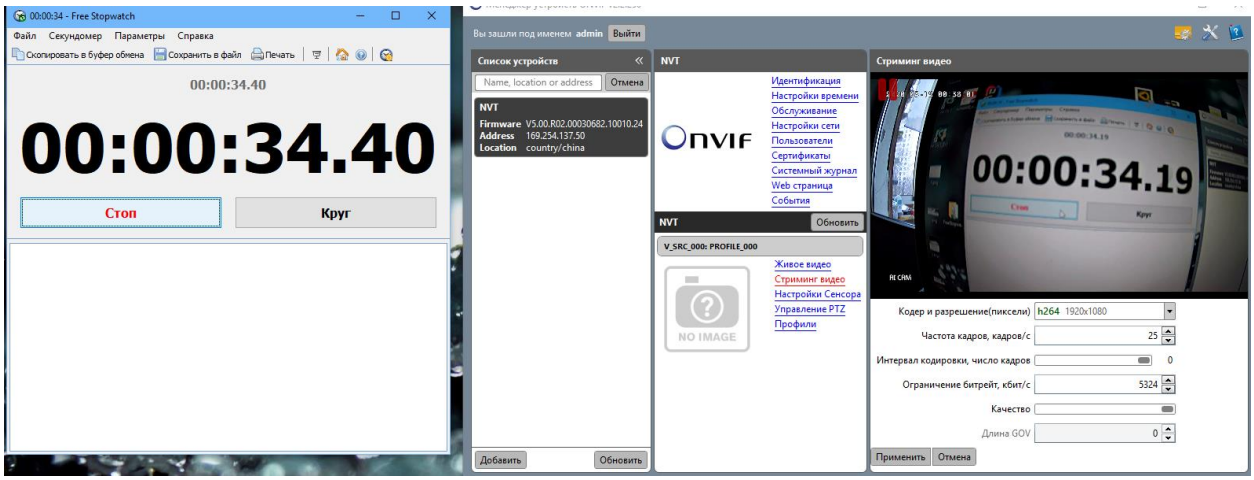
Замер 1


00:00:04 - Free Stopwatch
Файл Секундомер Параметры Справка
Скопировать в буфер обмена Сохранить в файл Печать
00:00:04.18
00:00:04.18
Стоп
Круг
Менеджер устройств ONVIF v2.2.250
Вы зашли под именем admin Выйти
Список устройств
Name, location or address
Отмена
NVT
Firmware V5.00.R02.00030682.10010.24
Address 169.254.137.50
Location country/china
Добавить
Обновить
NVT
Идентификация
Настройки времени
Обслуживание
Настройки сети
Пользователи
Сертификаты
Системный журнал
Web страница
События
NVT
Обновить
V_SRC_000: PROFILE_000
NO IMAGE
Живое видео
Стриминг видео
Настройки Сенсора
Управление PTZ
Профили
Стриминг видео
Кодер и разрешение(пиксели) h264 1920x1080
Частота кадров, кадров/с 25
Интервал кодировки, число кадров 0
Ограничение битрейт, кбит/с 5324
Качество
Длина GOV 0
Применить
Отмена

Замер 2



Замер 3



Задержка от 160 до 210 мс

4. Измерение потоков и задержек IP-камеры Beward BD3595Z33

Измерение потоков и задержек IP-камеры Beward BD3595Z33 производилось при двух разрешениях: 2048 на 1536 пикселей и 1920 на 1080 пикселей. Подключение выполнялось по протоколу RTSP, параметры кодека, частота кадров и битрейт показаны на рисунке 7.

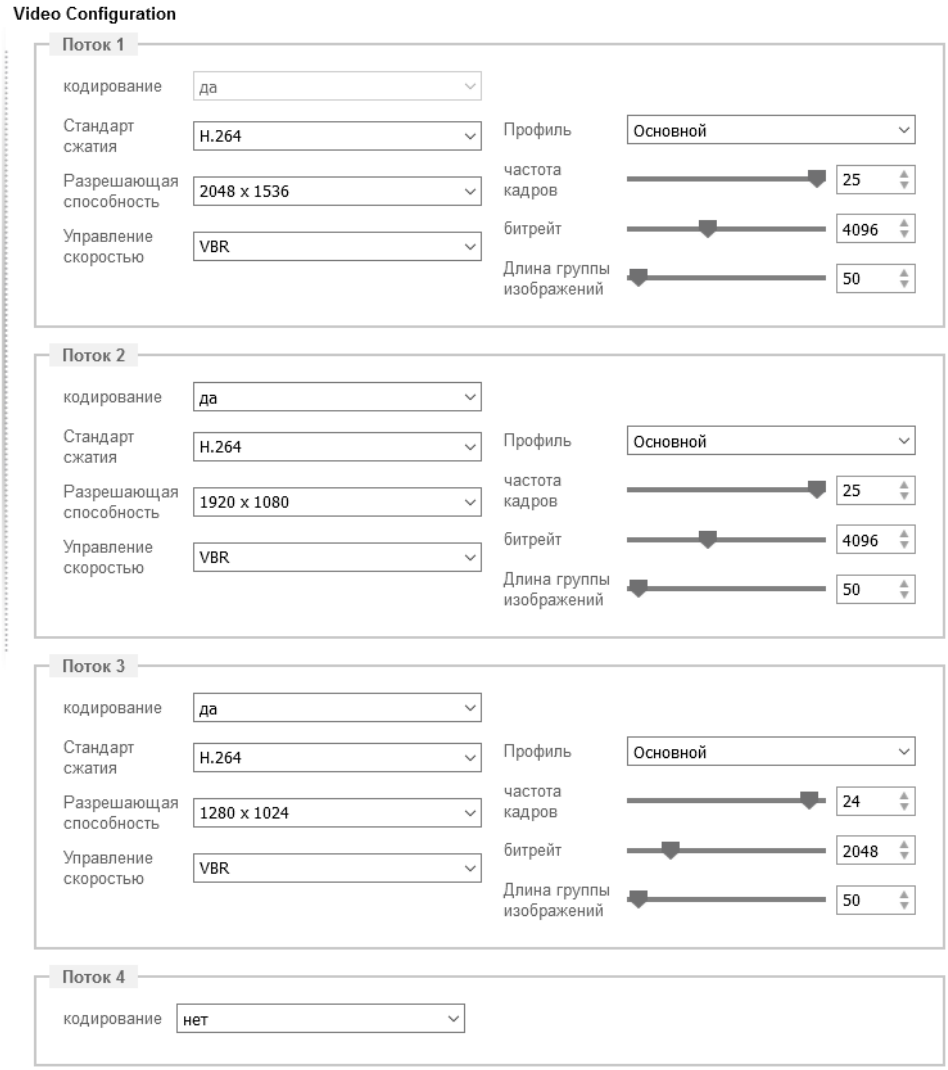


Рисунок 7 – Параметры передачи видео с камеры Beward BD3595Z33

Процесс измерения потоков и задержек представлен на рисунках 8 – 13.

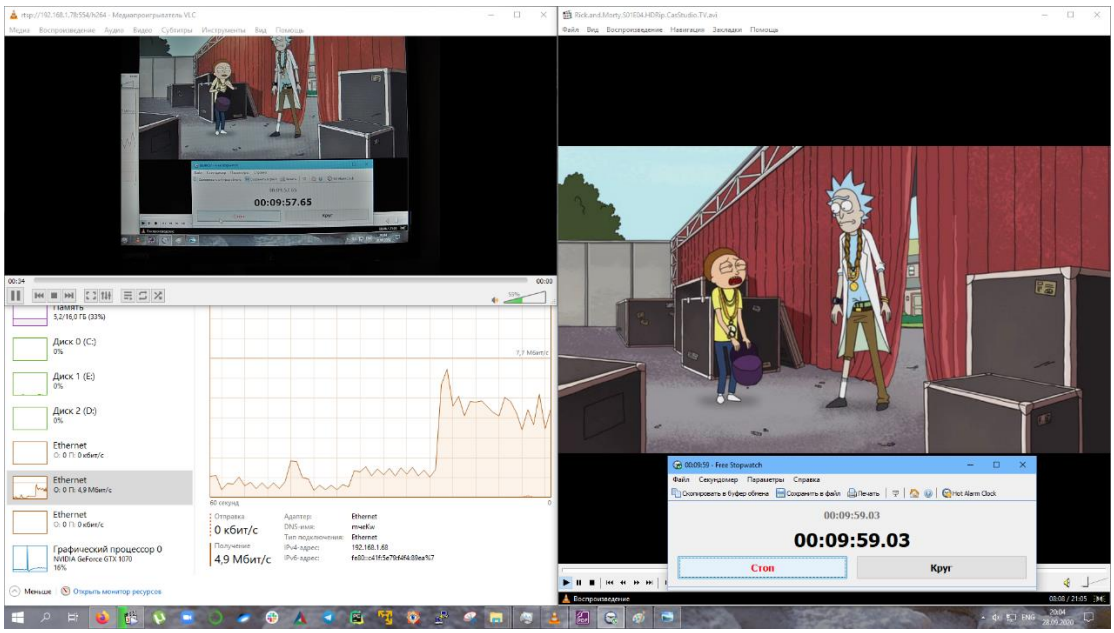


Рисунок 8 – Измерение 1, 1536p

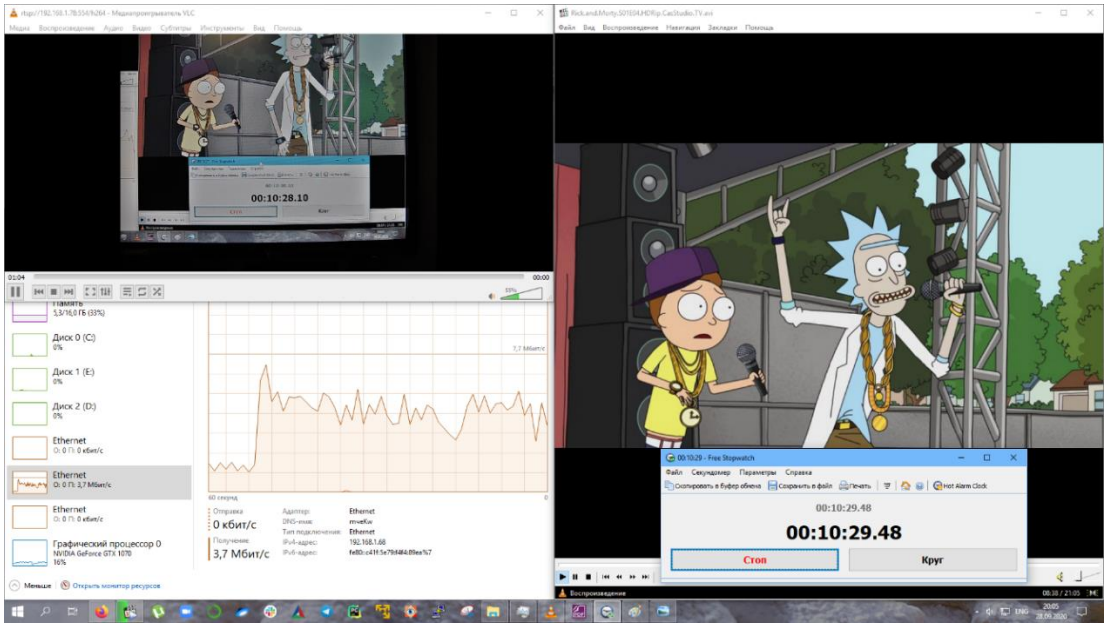


Рисунок 9 – Измерение 2, 1536p

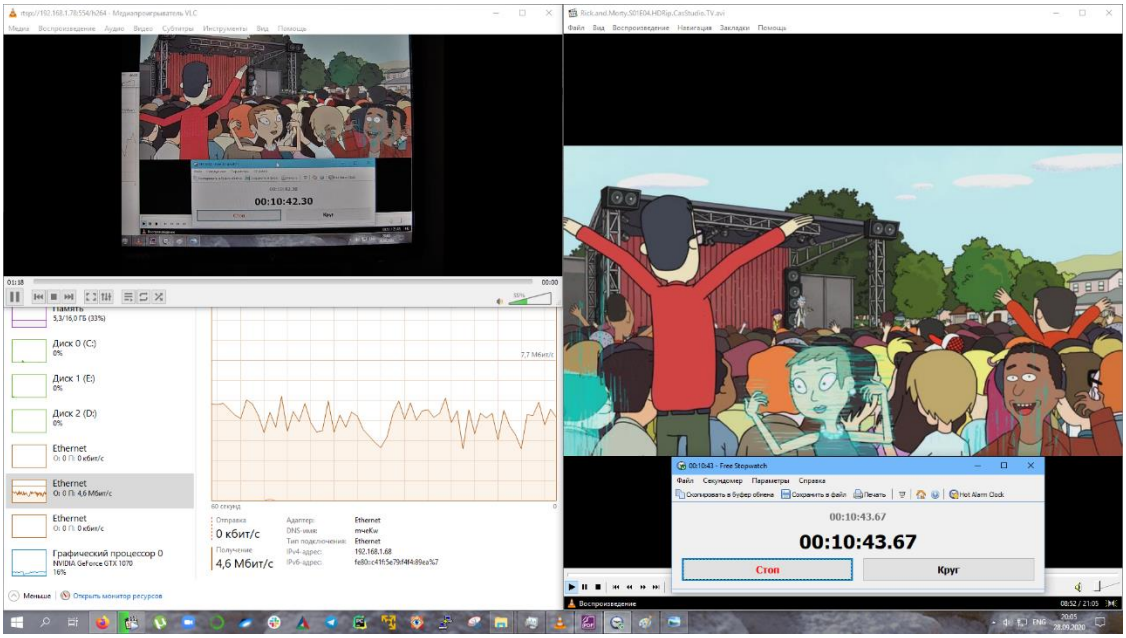


Рисунок 10 – Измерение 3, 1536p

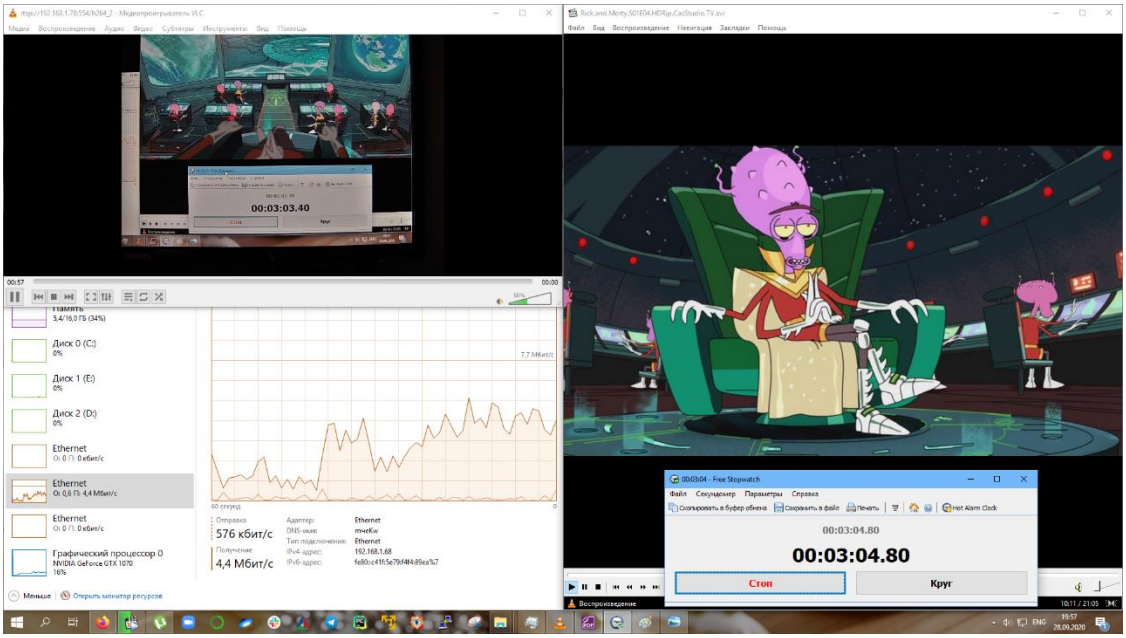


Рисунок 11 – Измерение 4, 1080p

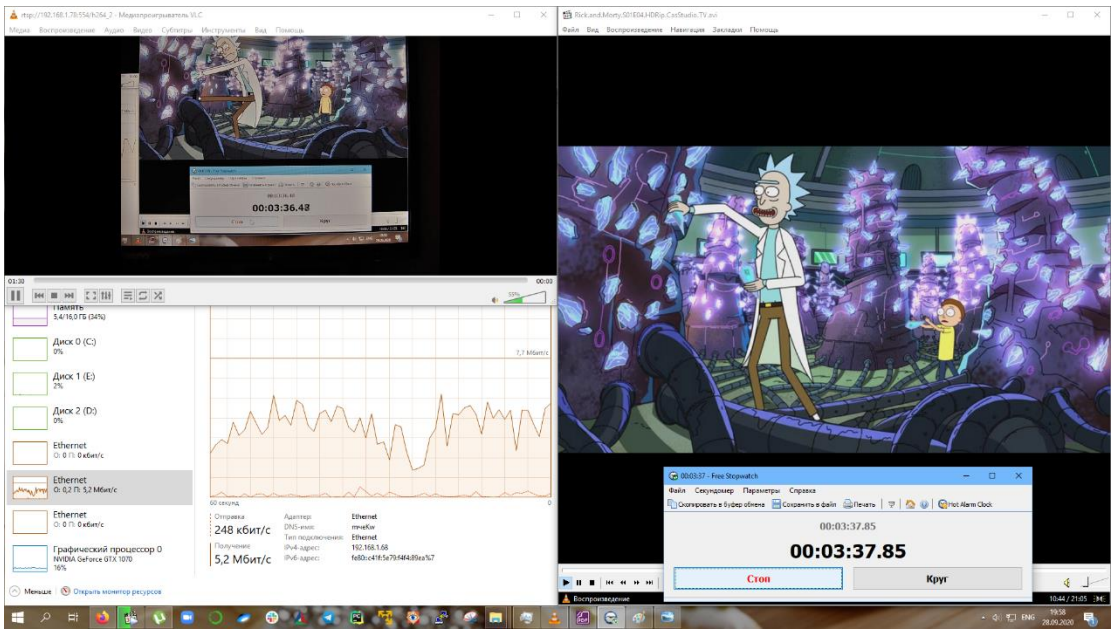


Рисунок 12 – Измерение 5, 1080p

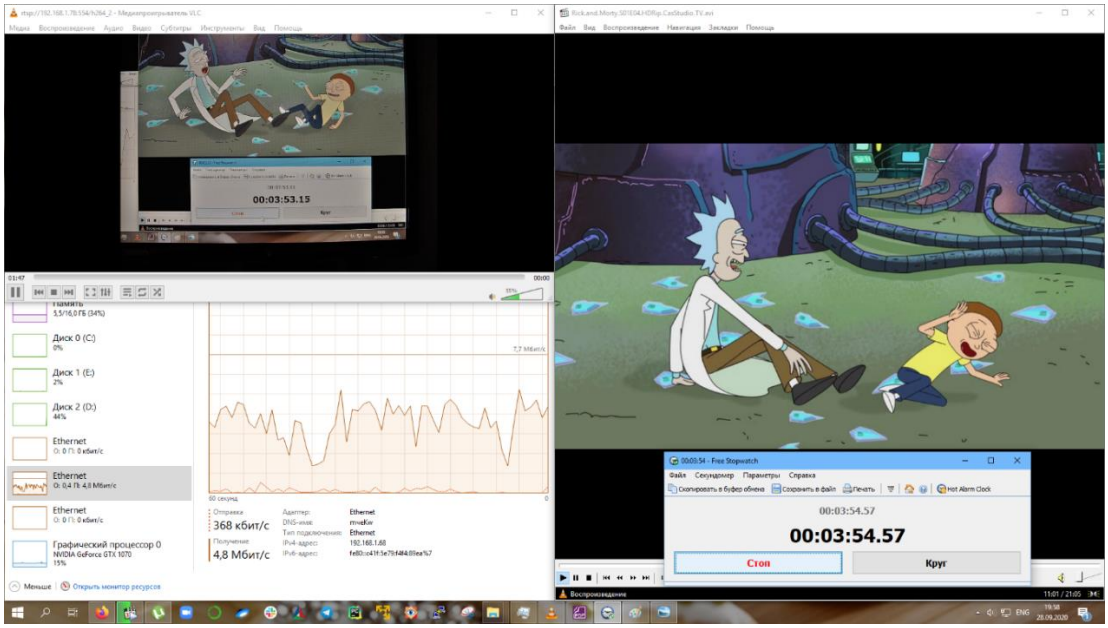


Рисунок 13 – Измерение 6, 1080p

С учётом кеширования. равного 1000 мс по умолчанию, задержки на максимальном разрешении составили в среднем 380 мс, величина потока колебалась в пределах от 3,7 Мбит/с до 4,7 Мбит/с. В простое поток составлял

около 800 кбит/с. При разрешении FullHD задержка составила 400 мс, а поток колебался от 4,4 Мбит/с до 5,2 Мбит/с.

ЗАКЛЮЧЕНИЕ

По итогам выполнения научно-исследовательской практики были изучены аглоритмы коррекции изображений с неравномерной освещённостью, проведено сравнение их производительности, а также измерены потоки и задержки IP-камеры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Теория ретинекса [электроный ресурс] – URL: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%80%D0%B5%D1%82%D0%B8%D0%BD%D0%B5%D0%BA%D1%81%D0%B0 (дата обращения: 31.08.2020).
2. Статья «Multiscale retinex» в журнале IPOL [электронный ресурс] – URL: http://www.ipol.im/pub/art/2014/107/ (дата обращения: 31.08.2020).
3. Цветовая модель HSV [электронный ресурс] – URL: https://ru.wikipedia.org/wiki/HSV_(%D1%86%D0%B2%D0%B5%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) (дата обращения: 31.08.2020).
4. Страница продукта [электронный ресурс] – URL:https://www.tinko.ru/catalog/product/287442/?ncc=1 (дата обращения: 20.07.2020).
5.

Приложение А. Листинг программы, использующей алгоритм SSR

```
import sys
import os

import cv2
import json
import numpy as np
import retinex
from pprint import pprint

with open('config.json', 'r') as f:
    config = json.load(f)

img = cv2.imread('./data/input_0_sel.png')

hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
hue, sat, val = cv2.split(hsv)

val_float = np.float64(val) + 1.0
val_float = retinex.singleScaleRetinex(val_float, 80)
val_float = (val_float - np.min(val_float)) / \
            (np.max(val_float) - np.min(val_float)) * \
            255

val_float = np.uint8(np.minimum(np.maximum(val_float, 0), 255))

print("HSV shape: ", len(hsv))
#pprint(hsv)
print(hue.size)
print(sat.size)
print(val.size)
print(val_float.size)

#print(hsv[..., 0].depth())
#print(hsv[..., 1].depth())
#print(hsv_2nd_ch.depth())

final_hsv = cv2.merge([hue, sat, val_float])
final = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)

cv2.imshow('orig', img)
cv2.imshow('hsv', hsv)
cv2.imshow('hsv[2] - brightness', val)
cv2.imshow('final hsv', final_hsv)
cv2.imshow('Final', final)

cv2.imwrite('hsv-brightness.png', val)
cv2.imwrite('hsv-corrected.png', final_hsv)
cv2.imwrite('brightness-corrected.png', final)

img_combined = np.hstack((cv2.cvtColor(val, cv2.COLOR_GRAY2BGR), final_hsv, final))
cv2.imwrite("three-in-one-hsv.png", img_combined)

cv2.waitKey(0)
```

Приложение Б. Листинг 2. Алгоритм MSR и его вариации

Файл `retinex.py`

```
import numpy as np
import cv2
from numba import jit


#@jit(nopython=True, parallel=True)
def singleScaleRetinex(img, sigma):
    retinex = np.log10(img) - np.log10(cv2.GaussianBlur(img, (0, 0), sigma))
    return retinex


# @jit(nopython=True, parallel=True)
def multiScaleRetinex(img, sigma_list):

    retinex = np.zeros_like(img)
    for sigma in sigma_list:
        retinex += singleScaleRetinex(img, sigma)

    retinex = retinex / len(sigma_list)
    return retinex


def colorRestoration(img, alpha, beta):
    img_sum = np.sum(img, axis=2, keepdims=True)
    color_restoration = beta * (np.log10(alpha * img) - np.log10(img_sum))
    return color_restoration


#@jit(nopython=True)
def simplestColorBalance(img, low_clip, high_clip):

    total = img.shape[0] * img.shape[1]
    for i in range(img.shape[2]):
        unique, counts = np.unique(img[:, :, i], return_counts=True)
        current = 0
        for u, c in zip(unique, counts):
            if float(current) / total < low_clip:
                low_val = u
            if float(current) / total < high_clip:
                high_val = u
            current += c

        img[:, :, i] = np.maximum(np.minimum(img[:, :, i], high_val), low_val)
    return img


def MSRCR(img, sigma_list, G, b, alpha, beta, low_clip, high_clip):

    img = np.float64(img) + 1.0

    img_retinex = multiScaleRetinex(img, sigma_list)
    img_color = colorRestoration(img, alpha, beta)
    img_msrcr = G * (img_retinex * img_color + b)

    for i in range(img_msrcr.shape[2]):
```

```
        img_msrcr[:, :, i] = (img_msrcr[:, :, i] - np.min(img_msrcr[:, :, i])) / \
                             (np.max(img_msrcr[:, :, i]) - np.min(img_msrcr[:, :, i])) * \
                             255

    img_msrcr = np.uint8(np.minimum(np.maximum(img_msrcr, 0), 255))
    img_msrcr = simplestColorBalance(img_msrcr, low_clip, high_clip)
    return img_msrcr


def automatedMSRCR(img, sigma_list):

    img = np.float64(img) + 1.0
    img_retinex = multiScaleRetinex(img, sigma_list)

    for i in range(img_retinex.shape[2]):
        unique, count = np.unique(np.int32(img_retinex[:, :, i] * 100),
return_counts=True)
        for u, c in zip(unique, count):
            if u == 0:
                zero_count = c
                break

        low_val = unique[0] / 100.0
        high_val = unique[-1] / 100.0
        for u, c in zip(unique, count):
            if u < 0 and c < zero_count * 0.1:
                low_val = u / 100.0
            if u > 0 and c < zero_count * 0.1:
                high_val = u / 100.0
                break

        img_retinex[:, :, i] = np.maximum(np.minimum(img_retinex[:, :, i], high_val),
low_val)

        img_retinex[:, :, i] = (img_retinex[:, :, i] - np.min(img_retinex[:, :, i])) / \
                               (np.max(img_retinex[:, :, i]) - np.min(img_retinex[:, :,
i])) \
                               * 255

    img_retinex = np.uint8(img_retinex)
    return img_retinex


#@jit(parallel=True)
def MSRCP(img, sigma_list, low_clip, high_clip):

    img = np.float64(img) + 1.0

    intensity = np.sum(img, axis=2) / img.shape[2]

    retinex = multiScaleRetinex(intensity, sigma_list)

    intensity = np.expand_dims(intensity, 2)
    retinex = np.expand_dims(retinex, 2)

    intensity1 = simplestColorBalance(retinex, low_clip, high_clip)

    intensity1 = (intensity1 - np.min(intensity1)) / \
                 (np.max(intensity1) - np.min(intensity1)) * \
                 255.0 + 1.0
```

```
    img_msrcp = np.zeros_like(img)

    for y in range(img_msrcp.shape[0]):
        for x in range(img_msrcp.shape[1]):
            B = np.max(img[y, x])
            A = np.minimum(256.0 / B, intensity1[y, x, 0] / intensity[y, x, 0])
            img_msrcp[y, x, 0] = A * img[y, x, 0]
            img_msrcp[y, x, 1] = A * img[y, x, 1]
            img_msrcp[y, x, 2] = A * img[y, x, 2]

    img_msrcp = np.uint8(img_msrcp - 1.0)
    return img_msrcp
```

Файл `main.py`

```
import sys
import os

import cv2
import json
import numpy as np
import retinex

data_path = 'data'
img_list = os.listdir(data_path)
if len(img_list) == 0:
    print('Data directory is empty.')
    exit()

with open('config.json', 'r') as f:
    config = json.load(f)

for img_name in img_list:
    if img_name == '.gitkeep':
        continue

    img = cv2.imread(os.path.join(data_path, img_name))

    img_msrcr = retinex.MSRCR(
        img,
        config['sigma_list'],
        config['G'],
        config['b'],
        config['alpha'],
        config['beta'],
        config['low_clip'],
        config['high_clip']
    )

    img_amsrcr = retinex.automatedMSRCR(
        img,
        config['sigma_list']
    )

    img_msrcp = retinex.MSRCP(
        img,
```

```
    config['sigma_list'],
    config['low_clip'],
    config['high_clip']
)

shape = img.shape
cv2.imshow('Image', img)
cv2.imshow('retinex', img_msrcr)
cv2.imshow('Automated retinex', img_amsrcr)
cv2.imshow('MSRCP', img_msrcp)

img_combined = np.hstack((img_msrcr, img_amsrcr, img_msrcp))
cv2.imwrite("three-in-one.png", img_combined)
cv2.waitKey()
```

Приложение В. Скрипт, реализующий многопроцессорность

```
import numpy as np
import cv2
import json
import retinex
import time
from multiprocessing import Process, Manager

"""
Multiprocessing conversion with single scale retinex
"""

def process_image(fr_list, process_no, ret_map):
    #global frame_no

    for frame_no, frame in enumerate(fr_list):

        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        hue, sat, val = cv2.split(hsv)

        val_float = np.float64(val) + 1.0
        val_float = retinex.singleScaleRetinex(val_float, 80)
        val_float = (val_float - np.min(val_float)) / \
                    (np.max(val_float) - np.min(val_float)) * \
                    255

        val_float = np.uint8(np.minimum(np.maximum(val_float, 0), 255))
        final_hsv = cv2.merge([hue, sat, val_float])
        final = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)

        print("Processing frame No: {:04d} in process No: {}".format(frame_no,
process_no))
        #out.write(final)
        #cv2.imshow("Before", frame)

        fr_list[frame_no] = final
        #cv2.imshow("After", fr_list[frame_no])
        #cv2.waitKey(0)
        frame_no += 1
    ret_map[str(process_no)] = fr_list


def main():
    with open('config.json', 'r') as f:
        config = json.load(f)

    cap = cv2.VideoCapture('./video/retinex-sample3.mp4')
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('output-hsv-mp.avi', fourcc, 30.0, (1280, 720))

    # frame_no = 0
    start = time.time()
    print("Начало преобразования в ", start)

    frames_list = []
    #frames_list_1 = []
    #frames_list_2 = []
```

```
#frames_list_3 = []
#frames_list_4 = []

"""
400 кадров для 4 потоков = 100 кадров для одного потока
"""
frames_counter = 400
# process_1_frame_no = 1
# process_2_frame_no = 1
# process_3_frame_no = 1
# process_4_frame_no = 1

while cap.isOpened() and frames_counter:
    ret, frame = cap.read()
    frames_list.append(frame)
    frames_counter -= 1

"""
for i in range(0, 400, 4):
    frames_list_1.append(frames_list[i])
    frames_list_2.append(frames_list[i+1])
    frames_list_3.append(frames_list[i+2])
    frames_list_4.append(frames_list[i+3])
"""

frames_list_1 = frames_list[0:100]
frames_list_2 = frames_list[100:200]
frames_list_3 = frames_list[200:300]
frames_list_4 = frames_list[300:400]

print("Разбивка на списки произведена")

hash_map = Manager().dict()

p1 = Process(target=process_image, args=(frames_list_1, 1, hash_map))
p2 = Process(target=process_image, args=(frames_list_2, 2, hash_map))
p3 = Process(target=process_image, args=(frames_list_3, 3, hash_map))
p4 = Process(target=process_image, args=(frames_list_4, 4, hash_map))

#cv2.imshow("Before", frames_list_1[0])

p1.start()
p2.start()
p3.start()
p4.start()

#cv2.imshow("After", frames_list_1[0])
#cv2.waitKey(0)

p1.join()
p2.join()
p3.join()
p4.join()

end = time.time()
print("Конвертация завершена в ", end)
print(end - start)

"""
    frame_list_final = frames_list_1
```

```
            frame_list_final.extend(frames_list_2)
            frame_list_final.extend(frames_list_3)
            frame_list_final.extend(frames_list_4)
        """
        frame_list_final = hash_map['1']
        frame_list_final.extend(hash_map['2'])
        frame_list_final.extend(hash_map['3'])
        frame_list_final.extend(hash_map['4'])

        print("len(frame_list_final)", len(frame_list_final))
        for item in frame_list_final:
            out.write(item)

        # Release everything if job is finished
        cap.release()
        out.release()
        cv2.destroyAllWindows()


if __name__ == '__main__':
    main()
```